HW1

Ben Cohen

March 13, 2013

1.5

I don't believe that you can apply the same engineering approach to all of the Software Application Domains listed in section 1.1.2. As example, Engineering/Scientific software which contains only "number crunching" algorithms take very little planning, as well as very little client interaction. The problems are very well defined and the main challenge is implementation and not catering to a client's expectations. Additionally, the levels of documentation, polish, and user friendlyness, and usability also vary vastly between these different types of software. For example, systems softwawre is often "rough" looking, with little to no GUIs, and web-apps must look flashy to the end user. In short, no, you cannot pick one general approach to engineering all kinds of software.

2.7

1) A short-term startup project. I think that the waterfall model would work well here. There is very little collaboration needed as often the developers are the people with the ideas. In addition, the short term nature of the project lends well to the rapid decent through the stages.
2) Anything "number crunching" related. The requirements for these are generally very well defined, and the challenges lie in the implementation. Once the problem is outling though, very little communication is required.
3) A project that has a very specific budget and delivery date. The waterfall method woul dbe good for ensuring that you progress at a steady rate and don't go back too far in the process.

2.10

1) Projects with multiple deliverables. I think that the incremental method would be good for this as the regular waterfall method is good for single release projects, so using the incremental model for something with multiple releases is just like using the waterfall model numerous times.
2) A very large project which has to be precise, like an airplane. This method allows for lots of testing and communication to be sure that everything is perfect on release.
3) Projects with a dynamic group of stakeholders, and ones where requirements change often. For example if you're developing software for a company that's undergoing an overhaul or one that's about to be bought out. In these situations the client's needs might drastically shift.

3.2

Agility for software projects means being able to constantly adapt to new needs, requirements, deadlines, etc. It means working in dynamic teams and being well organized, while maintaining constant communication in between teams and clients.

3.3

Having an iterative process makes it easier to constantly change because there are mutiple points throughout the development process for gathering requirements and planning. All of the discussed processies are iterative to a degree as they all have multiple points to adapt to a change in requirements. It is possible to complete a project in one iteration and still be relatively agile, as you can still use teams and have good communication and adaptation within the teams, even if you only do everything once.

4.1

Absolutely. Agility does not mean sacrificing quality, in fact it means the exact opposite. Agility stresses many things in an effort to make the best project. It wouldn't be used if quality was sacrificed. An agile strategy takes time into account and tries to use the alloted time most efficiently to produce the best possible product.

4.3

Seperation of concerns means breaking up a large task/program into smaller tasks/modules. It entails having all of the main concerns divided up and dealt with individually.

5.1

I think that this is because many people have the "let's just jump in and code" philosophy. It's much more fun to start coding than it is to plan, and often it seems like planning is trivial, so people often skip it. I personally don't think you can every skip requirements getting and planning, no matter how trivial it seems.

5.9

Use case for using a charge card for a meal. Request Bill, get bill, check bill, give waiter your card, waiter swipes card, request is sent to bank, bank echoes back confirmation, reciept prints, waiter brings reciept back to customer, customer optionally adds tip, restaraunt later rings full ammount back to bank.